2015/2016

Relatório de Trabalho de SO1

Discentes:

- Hiago Oliveira 29248

- Luís Sousa 32095

- Gil Catarino 32378

Docente:

- Nuno Miranda

Introdução

Nesta cadeira foi-nos pedido para implementar um escalonador de processos sobre o modelo de 5 estados (new, ready, blocked, run, exit) dado nas aulas de Sistemas Operativos 1.

Main

O programa começa por chamar uma função que recebe o nome de um ficheiro e este irá ler o ficheiro e transformar cada linha lida num processo (PCB), que por sua ver irá conter uma lista de instruções para depois carrega-las em memória no momento em que entra no sistema.

O programa segue depois fazendo o escalonamento dos processos lidos do ficheiro. O flow do programa está explicado mais abaixo num diagrama.

Fez-se a implementação com queues (linked list) para facilitar a transição de processos entre estados, visto ser mais fácil do que trabalhar com arrays (em C).

No início do ficheiro estão as opções do escalonador, nomeadamente a quantidade de memória disponível, o quantum, o tip de fit a usar, etc.

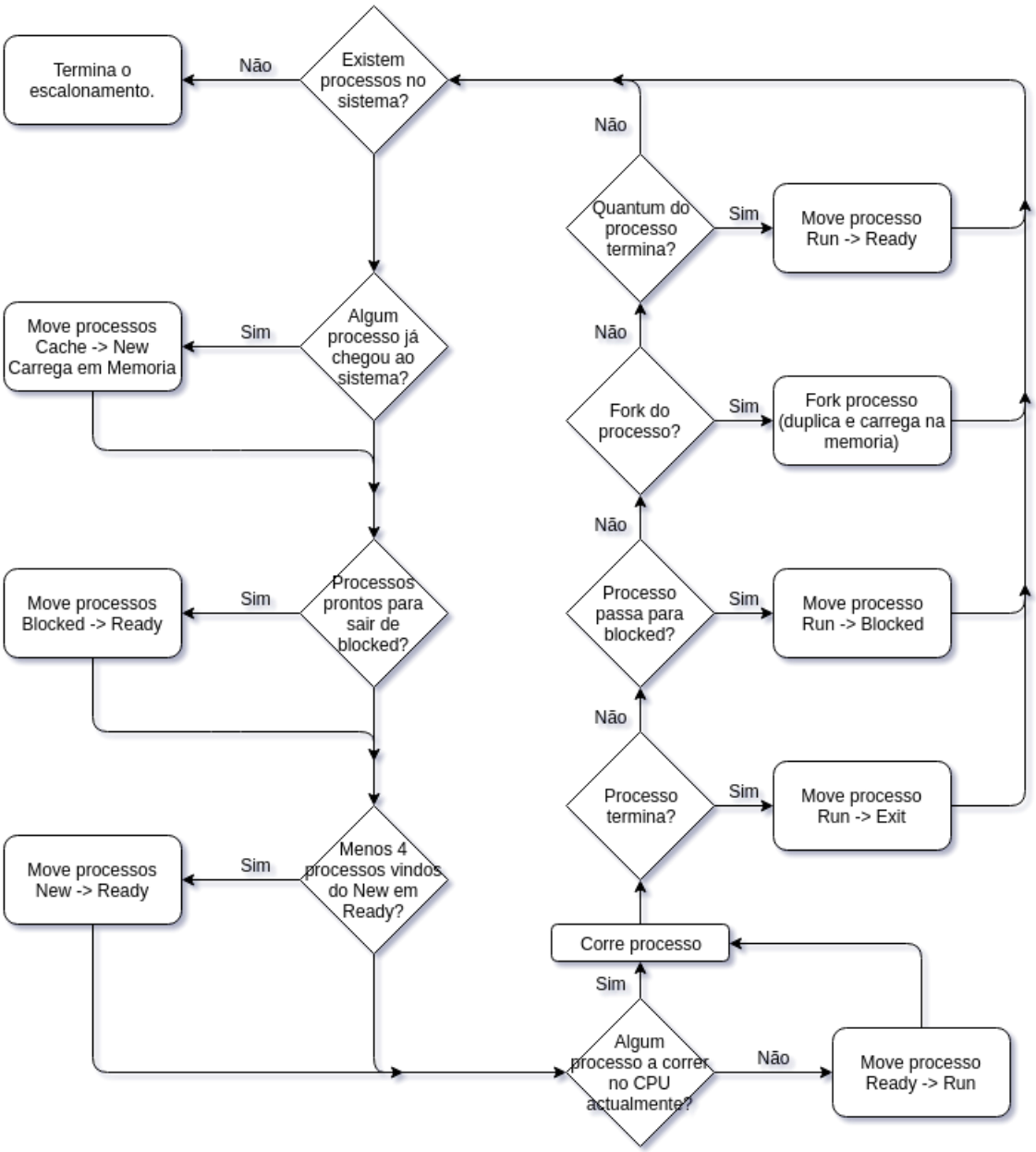
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "scheduler.h"
#define QUANTUM 4
#define MAXNEWREADY 4
#define MEMSPACE 300
#define TYPEFIT 1    // Best fit 1, Worst fit
#define DEBUGPRINT 0 // Normal 0, Debug Print
// TODO NUM CICLO FORK; ESTRUTURA GERIR MEM; P
int read_input(char *file_name, state **cache)
{
```

```c
typedef struct node {
    void *val;
    struct node *next;
} node_t;

node_t *new_list();
void add_start(node_t **head, void *val);
void add_end(node_t **head, void *val);
void *remove_first(node_t **head);
void *remove_last(node_t **head);
void *getelement(node_t *head, int n);
void clear_list(node_t **head);
```

Criou-se também as estruturas do pcb, e do estado para ter as informações necessárias para cada processo.

```c
typedef struct pcb {
    int pid;        // process id
    int at;         // arrive time
    int mem_ptr;
    int pc;
    int cb;         // cicles blocked
    int fn;         // process from new
    int num_inst;
    node_t *instruction_list;
} pcb;

typedef struct state {
    int num_processes;
    node_t *process_list;
} state;
```

Comportamento do programa (diagrama)



Através de um Switch foi implementado a tabela de instruções, que para os 10 casos possíveis com o código e a variável a usar.

```
int run_process(void **mem, pcb **p)
{
    inst *it = (inst*)mem[((*p)->mem_ptr + (*p)->pc)];
    int opcode = it->code;
    int v = it->var;
    int *destination_var = (int*)mem[((*p)->mem_ptr) - (10 - v)];
    int state_transition = 0;

    switch (opcode) {
        case 0:
            *destination_var = 0;
            (*p)->pc += 1;
            break;
        case 1:
            *destination_var += 1;
            (*p)->pc += 1;
            break;
        case 2:
            *destination_var -= 1;
            (*p)->pc += 1;
            break;
        case 3:
            if (*destination_var == 0) {
                (*p)->pc += 1;
            } else {
                (*p)->pc += 2;
            }
            break;
```

…

Tabela de instruções:

Códigos	Instruções	Significado
0X	ZERO X	X = 0
1X	ADD X	X ++
2X	SUB X	X --
3X	IF X	If X ==0, goto next line (PC++); else goto next line +1 (PC += 2)
4Y	BACK Y	Goto PC -= Y, em que Y é logo o valor do salto, não se vai consultar o valor da variável
5Y	FORW Y	Goto PC += Y, em que Y é logo o valor do salto, não se vai consultar o valor da variável
6X (facultativa)	FORK X	X = Fork() (facultativa)
7X	DISK SAVE X	Wait…
8X	COPY X	X0 = X
9X	EXIT	Exit

Best Fit - Este permite encontrar o bloco livre de memória com o tamanho mais próximo do bloco necessário para o processo.

Wors Fit – Faz o mesmo que o Best Fit mas com o bloco de maior tamanho.

Também foi implementado neste trabalho o Fork X(6)

Fork – duplica o processo com as instruções a partir da instrução seguinte ao fork do processo original.